AFRL-AFOSR-JP-TR-2017-0046

Towards a Better Distributed Framework for Learning Big Data

Shou-de Lin
NATIONAL TAIWAN UNIVERSITY

06/14/2017
Final Report

Air Force Research Laboratory
AF Office Of Scientific Research (AFOSR)/ IOA
Arlington, Virginia 22203
Air Force Materiel Command

REPORT DOCUMENTATION PAGE	*Form Approved* *OMB No. 0704-0188*

1. REPORT DATE *(DD-MM-YYYY)* 14-06-2017	**2. REPORT TYPE** Final	**3. DATES COVERED** *(From - To)* 14 May 2015 to 13 May 2017

4. TITLE AND SUBTITLE Towards a Better Distributed Framework for Learning Big Data	**5a. CONTRACT NUMBER**
	5b. GRANT NUMBER FA2386-15-1-4013
	5c. PROGRAM ELEMENT NUMBER 61102F
6. AUTHOR(S) Shou-de Lin	**5d. PROJECT NUMBER**
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) NATIONAL TAIWAN UNIVERSITY 1, ROOSEVELT RD., SEC. 4 TAIPEI CITY, 10617 TW	**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) AOARD UNIT 45002 APO AP 96338-5002	**10. SPONSOR/MONITOR'S ACRONYM(S)** AFRL/AFOSR IOA
	11. SPONSOR/MONITOR'S REPORT NUMBER(S) AFRL-AFOSR-JP-TR-2017-0046

12. DISTRIBUTION/AVAILABILITY STATEMENT
A DISTRIBUTION UNLIMITED: PB Public Release

13. SUPPLEMENTARY NOTES

14. ABSTRACT

This work aimed at solving issues in distributed machine learning. The PI's team proposed three directions to work on. First, they designed solutions to speed up the alternating direction method of multipliers (ADMM) for distributed data. Second, they focused on a client-server learning scenario in which an online, semi-supervised learning approach is designed to reduce the communication load. Finally, the team proposed the parallel least-squares policy iteration (parallel LSPI) to parallelize a reinforcement policy learning.

15. SUBJECT TERMS
Machine Learning

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON**
a. REPORT	**b. ABSTRACT**	**c. THIS PAGE**			HONG, SENG
Unclassified	Unclassified	Unclassified	SAR	27	**19b. TELEPHONE NUMBER** *(Include area code)* 315-229-3519

Standard Form 298 (Rev. 8/98)
Prescribed by ANSI Std. Z39.18

Final Report for AOARD Grant FA2386-15-1-4013

Towards a Better Distributed Framework for Learning Big Data

05/11/2017

Name of Principal Investigators (PI and Co-PIs): Shou-De Lin

- e-mail address : sdlin@csie.ntu.edu.tw
- Institution : National Taiwan University
- Mailing Address : CSIE Department, National Taiwan University, Taipei, Taiwan
- Phone : +886-2-33664888-333
- Fax : +886-2-33664898

Period of Performance: 05/14/2015 – 05/13/2017

Abstract: This work aims at solving issues in distributed machine learning. We propose three directions to work on. First, we design solutions to speed up the alternating direction method of multipliers (ADMM) for distributed data. Second, we focus on a client-server learning scenario in which an online, semi-supervised learning approach is designed to reduce the communication load. Finally, we propose the parallel least-squares policy iteration (parallel LSPI) to parallelize a reinforcement policy learning.

Introduction:

Speeding up ADMM

We consider training a classifier given large data that are distributed. Since data are distributed on different machines and are too big to relocated, an efficient way to train a classifier is to develop a distributed optimization method. The alternating direction method of multipliers (ADMM) can be exploited to solve this problem, in which each local machine updates its learned model and the master machine tries to reach a consensus. Specifically, the setting we consider is that there are g machines with each machine m learns a model w_m from its local data N_m. The learned models w_m are required to agree with each other to form a consensus c. Mathematically, we are interested in solving the following linear classification problem,

$$minimize\_\{w_1, w_2, \ldots, w_g, c\} \sum_{m=1}^{g} \sum_{i \in N_m} C \max{(1 - l_i\, w_m'\, z_i\, ,0)^2} + \frac{1}{2} c' c \quad (1)$$

$$\mathrm{s.t.}\, \mathrm{w_m} - c = 0$$

where l_i and z_i is a label/feature pair of the i sample on machine m, and C controls the regularization effect. Although squared hinge loss is shown here, the objective is in fact general and other types of loss function can be utilized. For example, if we use squared loss, the objective becomes a regression problem.

There exist some related works using ADMM for distributed data. For example, [Zhang et. al. 2012] have implemented ADMM to solve it. The update of w at an iteration of ADMM turns out to be similar to standard SVM formulation. Thus, [Zhang et. al. 2012] adapted the approach, stochastic dual coordinate assent, which is implemented in a well-known LIBSVM toolkit to solve it. As the result, several ``inner'' iterations are required to obtain w_m at each iteration of ADMM. When the data on each local machine is also large, the computational cost for one pass of full data drastically increases, which makes the update of w_m consume a substantial amount of time. We aim at dealing with this issue and proposed two method.

The first approach is leveraging the idea from stochastic gradient descent method (SGD) [Bottou et al. 2010] for ADMM. SGD has shown its merits on solving large-scale optimization problem. At each iteration, it samples an instance to compute the gradient instead of using all the data as in the traditional gradient descent method. Several works has shown that SGD can solve large-scale problem more efficiently comparing to gradient descent [Bottou et al. 2010]. It is known for SGD that only a few samples are needed to achieve sufficient descent of the objective at the beginning. To achieve similar effect in ADMM, at the first few iterations, each local machine only uses a subset of its data instead of using all the data to update its model. For example, at the first iteration, each machine samples half of its data to compute w_m. Then, the sample size is increased for next iteration, and eventually each machine utilizes all of its local data. The method requires fewer computations at the first couple iterations, and therefore has the potential to achieve faster convergence. Since the method has cheaper iteration costs at the beginning, the method converges faster in terms of time. More importantly, we provide a theorem that guarantees the method to enjoy the same convergence rate in terms of the number of the iterations as the standard ADMM.

Following the idea of sampling subset of data instead of full to update the model before each round of communication, we convert the objective (1) to the dual domain and proposed our second approach that performs ADMM on the dual domain. As each dual variable corresponds to a sample, sampling a subset to update the model becomes easier and more natural than in primal domain. The algorithm for performing ADMM on the dual of (1) turns out to be equivalent to SDCA-ADMM [Suzuki 2014], which is originally proposed to solving objective with complex regularizations (e.g. group lasso [Jacob et al. 2009], graph guided

SVM [Ouyang et al. 2013] by combining ADMM and stochastic dual coordinate ascent (SDCA). However, these works do not consider generalizing the method on distributed data. We show the algorithm can solve for optimization on distributed data as well.

To summarize, our contribution are 1) proposing a simple, easy to implement, yet effective way to accelerate the ADMM on distributed data with a theoretical guarantee, 2) proposing running ADMM on the dual of the objective and showing the advantages of doing it, 3) showing the effectiveness of our methods on several datasets.

Communication-Efficient Online Semi-Supervised Learning in Client-Server Settings

This work considers such a setting where a set of distributed clients each generate an ongoing stream of data and a server seeks to learn a model of the data. We impose two practical limitations on the setting. First, because of the costs of having humans label large quantities of data, we assume that only a small fraction of the data are labeled. In particular, we focus on a setting where only the first, e.g., 2% of the training data are labeled. Second, because communication bandwidth is often expensive and battery-draining (e.g., a mobile device on a cellular network), we seek communication-efficient solutions such that each client is limited to sending to the server only a small fraction of the unlabeled data it generates, and limited in how much information it receives from the server.

An elegant solution to these problems will face many challenges. First, the amount of data generated by clients can be huge, and even potentially unlimited. As a result, the vast majority of data on the server are unlabeled. Typically, it is not sufficient to train a model with a good generalization ability based merely on limited labeled data. Second, when the volume and velocity of data is high, it is very costly and impossible to store all data either on clients or the server. Thus, traditional approaches that first store data and then train on a static collection are not appropriate in this case. Third, transmitting massive data on the network is discouraged in practice, especially when the network bandwidth is restricted or the communication cost is expensive (e.g., on a cellular network). It may also be mis-classified as a *denial-of-service attack*, and dropped/blocked.

By considering online, semi-supervised, and active learning jointly, our goal is to develop a modular framework for learning from a remote partially labeled data stream while reducing the bandwidth consumption. We present a novel framework for solving this learning problem in an effective and communication-efficient manner (see Figure 1). On the server side, our solution combines two diverse learners working collaboratively, yet in distinct roles, on the partially labeled data stream. A compact, online graph-based semi-supervised learner is used to predict labels for the unlabeled data arriving from the clients. Specifically, we adapt the Harmonic Solution learner to online use via an incremental k-center clustering approach that maintains the graph structure solely on a set of k centroid nodes. Random samples are then repeatedly drawn from the model according to the confidence of its prediction, and used to train a second learner on the server, a linear classifier (specifically, a soft confidence-weighted classifier). The second learner updates its hypothesis based on these samples and their predicted labels. We show how these two learners can be combined in an optimization problem. On the client side, our solution prioritizes data based on an active-learning metric that favors instances that are more uncertain (i.e., close to the classifier′s decision hyperplane) and yet far from each other (as measured by covariance). To reduce communication, the server sends the classifier′s weight-vector to the client only periodically. At any point in time, the classifier can be used as a standalone model for predicting labels for new test data.

Parallel least-squares policy iteration

Learning an optimal policy for MDPs with large state space has gained many interests recently. Different from previous works, our proposed method is inspired by the recent success in distributed optimization. The goal is to parallelize an existing policy iteration method called least-squares policy iteration. The algorithm takes advantage of the multi-core or multi-machine architecture, where each worker (one per core or machine) individually executes a fraction of episodes and estimates a parameter while a consen- sus is maintained by parameter averaging. With the feedback of global consensus, each worker can access the information learned by other workers at the previous iterations. As the result, the learning process of each individual worker can be accelerated, as compared to learning alone.
Our work aims at answering the following question: Given multiple computational resources, how to efficiently solve an MDP? In our problem, each worker faces the same MDP, and each worker communicates with others about the estimated parameter during learning. Thus, our work can be regarded as a complement to multi-agent MDP.

Our analysis on parallel LSPI shows that the correlation between the learning processes of each individually learned model can influence the effectiveness of the method. The com- putation gains achieved with parallel LSPI is less significant when there exists high correlation between workers. To deal with this issue, a heuristic is proposed to encourage each worker to explore (i.e. taking random action) more when it collects samples, which increases the randomness and in turn reduces correlation.

To summarize, we propose parallel LSPI to efficiently solve an MDP through parallel programming. Our method can also balance the communication overhead and required number of iterations to find the optimal solution, which is suitable for situation when only limited bandwidth is available. We give some analysis for the proposed method and conduct experiments to show its effectiveness on queueing networks and persistent search and track domains.

Method/Theory/Experiment:

Speeding up ADMM

We begin by describing how to apply ADMM in a distributed data problem. The augmented Lagrangian of objective (1) is

$$L_\rho(w,c,\lambda) = \sum_{m=1}^{g} \sum_{i \in N_m} C \max(1 - l_i w_m' z_i, 0)^2 + \frac{1}{2} c'c + \sum_{g=1}^{m} \frac{\rho}{2} (w_m - c)'(w_m - c) + \lambda_m'(w_m - c) \quad (2)$$

The algorithm in ADMM consists of

wk+1 = argmin_w $L_\rho(w,c,\lambda)$ (3)
c k+1 = argmin_c $L_\rho(w,c,\lambda)$ (4)
λk+1 = argmin_λ$L_\rho(w,c,\lambda)$ (5)

where k is the iteration index, w is { w1, …, wg} and and λ is { λ1,…, λg}. Note that solving (3) is similar to solving the objective of standard SVM. To see this, we rewrite it in explicit form.

$$w_m^{k+1} = \operatorname{argmin}_w \; C \sum_{i \in N_m} \max(1 - l_i w_m' z_i, 0)^2 + \frac{\rho}{2} (w_m - c^k + \lambda_m^k)'(w_m - c^k + \lambda_m^k) \quad (6)$$

Therefore, we can use an existing optimization methods for SVM for solving (6).

A sampling approach for fast ADMM on primal objective (1)

From the previous section, we know that ADMM requires solving the SVM-like problem (6) at every iteration of ADMM. Each iteration requires many inner iterations to complete. When local data is large, it will take a substantial amount of time. To deal with this, we propose a way to alleviate the training cost. At the earlier iterations, each local machine only uses a subset of its data instead of using all the data to update its learned model. As the algorithm continues, each machine gradually increases the amount of data used and finally reaches the full capacity. This method enjoys similar fast decrease of objective value as SGD does at the first few iterations and shares the same convergence rate in the long term as using the full dataset every iteration. Thus, for each iteration, each machine solves the following instead of (6) .

$$w_m^{k+1} = \mathrm{argmin}_w \; C \sum_{i \in N_m^k} \max(1 - l_i\, w_m' z_i, 0)^2 \; + \frac{\rho}{2}(w_m - c^k + \lambda_m^k)'(w_m - c^k + \lambda_m^k) \qquad (7)$$

where we have replaced N_m with N_m^k, which represents the amount of data used on machine m at the k_{th} iteration. The amount of data used at the k+1 iteration, N_m^{k+1}, satisfies

$$N_m^{\{k+1\}} = \max\{ N_m^k \times \beta, N_m \} \qquad (8)$$

where β>1 is the increasing rate. Note that N_m stands for all of the data of machine m. For example, we can initialize N_m^1 to 0.5 N_m, and set β to 1.1, meaning the amount of data used for training at each iteration in ADMM increases by 10 percents each iteration. The optimization procedure of the modified algorithm is roughly the same as the traditional one; it iterates over (3)-(5), except that the sub-problem (3) or (6) is replaced by (8), which can be solved in a similar manner as described in the previous section.

A sampling approach for fast ADMM on dual objective of (1)

We give another method that still follows the idea of sampling subset of data on each round of communication. This section begins by converting the primal form (1) to dual form. In order to make the dual form compact, we relax the constraint and approximate it. As each dual variable corresponds to a sample, sampling a subset to update the model becomes easier and natural. We then show how to integrate the sampling idea in performing ADMM on the dual of (1). Solving the dual of (1) by ADMM turns out to be equivalent to SDCA-ADMM. We then propose some techniques to efficiently perform SDCA-ADMM for distributed data.

Converting primal (1) to approximated dual form

To achieve the goal, we transform each feature vector from p-dimensional feature space to pg-dimensional features space. That is, the dimensions of augmented feature space is $p \times g$, which is g times larger than the original feature dimension p.

Let us denote the original feature vector of the i sample on the m machine as $zz_{\{i,m\}}$.

The new feature is

$z_{\{i,m\}}\,((m-1)p + 1{:}mp) = zz_{\{i,m\}}$

the other entries in $z_{\{i,m\}}$ are set to zeros. Thus, the data matrix Z would be a block diagonal matrix. Let us denote the diagonal blocks as $Z_{(m)}$, m ϵ {1,…,g}.

Note that $Z_{(m)}$ is the submatrix of Z_m that consists of original features $zz_{\{i,m\}}$.

We now turn to specify a matrix B that encodes the constraint in objective (1), which is

$w_1 = w_2 = \cdots = w_g$

Since the feature dimensions becomes $p \times g$ dimensions, so does the corresponding classifier w. Denote the subvector $w_{(m)}$ the m_{th} block of w. Due to the augmented features we design, we can view $w_{(m)}$ as a model learned by the local machine m. Then, we specify the transform B to be

$$B \in \mathbb{R}^{pg \times g} = \begin{pmatrix} 1_p & & & -1_p \\ -1_p & 1_p & & \\ & -1_p & 1_p & \\ \cdots & \cdots & \cdots & \cdots \\ & & -1_p & 1_p \end{pmatrix}$$

where 1_p means the p-dimensional vector of all 1's. Thus, $B^T w = 0$ is equal to the constraint $w_1 = w_2 = \cdots = w_g$ that encourages the model associated with each machine to agree with each other.

To make the dual form more compact and simplified, we relax the constraint and propose to use a regularization term Ψ so that $\Psi(B^T w)$ can have the similar effect of the constraint. We choose Ψ to be a squared of L2 norm. Thus, $\Psi(B^T w)$ would be

$$\frac{1}{2} c' \sum_{m=1}^{g} \left(w_m - w_{\{m+1\}}\right)' \left(w_m - w_{\{m+1\}}\right)$$

where $w_{g+1} = w_1$. The regularization penalizes the difference of a machine with its neighbors (in terms of the index m) and encourages the subvectors $w_{(m)}$ of w to agree with each other. Note that the specification of the transform B is flexible. If we have a prior about the relation between the models, we can easily encode it in the transform.

To summarize, the conversion to the approximated dual form is

minimize_w $\sum_{m=1}^{g} \sum_{i=1}^{n_m} f_{\{i,m\}} \left(z'_{\{i,m\}} w\right) + \Psi(B' w)$

= - minimize_{x,y}$\sum_{m=1}^{g} \sum_{i=1}^{n_m} f^*_{\{i,m\}}(z'_{\{i,m\}} w) + \Psi(B' w)$ s.t. Zx + By = 0 (11)

Here $f_{\{i,m\}}$ is the loss function associated to the sample i on machine m, and the symbol * is used to represent the corresponding conjugate function. Note that x and y here are called dual variables in the literature.

Exploiting diagoal structures of transformed feature space

Now we directly apply the standard ADMM (e.g. (3)~(5)) to solve the dual problem (11). But, since we have transfer the problem into the high dimensional feature space, the computations would be high. Yet, it turns out that we can leverage the structure of transformed feature space. In the following, we give an example about how to compute $Z' \times w$, where Z is the transformed data matrix in $R^{\{pg \times N\}}$ and w is the concatenated classifiers computed by each local machine whose dimensions is pg. This high-dimensional matrix-vector multiplication apppears in the ADMM updates. The direct computation causes high computation, large memory consumption, and very frequent communication.

Our example given is g=3 cases (i.e. data are distributed on three machines). Suppose I_1, I_2, and I_3 batches of dual coordinates are chosen at current iteration, so the current mini-batch I={I_1,I_2,I_3}. Since matrix Z is a block diagonal matrix and all the off-diagonal blocks are zeros (which are filled with the slash lines on the graph), the computations can be decomposed into smaller components $Z_{(1),I\_1}^T$ $w_{(1)}$, $Z_{(2),I\_2}^T$ $w_{(2)}$, and $Z_{(3),I_3}$T $w_{(3)}$, each is independently computed on the respective local machine. Thus, the unnecessary computation and communication can be avoided.

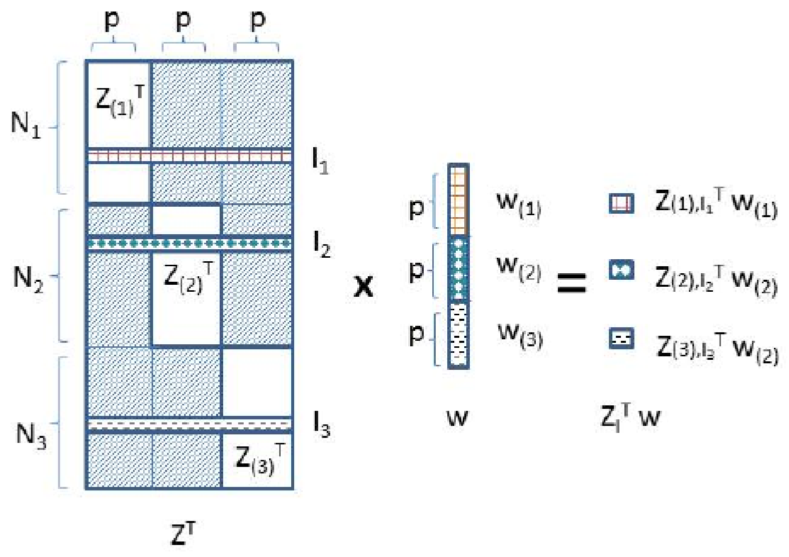


Fig. 1. An example to illustrate the way to leveraging the structure of transformed data matrix to improve the computing performance.

To summarize, our ADMM on the dual problem is shown in (11). The algorithm is a variant of ADMM applied on the dual domain which we have derived and the trick we desribed above. By variant, we mean that in each iteratoin, a batch of dual coordinates x are updated instead of all the coordinates, which is in the spirit of SDCA.

```
Algorithm 1 Distributed SDCA-ADMM
Input: parameters $\rho, \gamma, \eta_{Z,I}, \eta_B$
Initialize $x_0 = 0$, $y_0 = 0$, $w_0 = 0$
for $t = 1$ to $T$ do
    1. The master receives $c_{(m)}^{(t-1)}$ and update $y^{(t)}$.
    $q^{(t)} = y^{(t-1)} + \sum c_{(m)}^{(t-1)}$
    $y^{(t)} = q^{(t)} - prox(q^{(t)} | n\Psi(\rho\eta_B \cdot)/(\rho\eta_B))$
    2. The master broadcasts $y^{(t)}$.
    3. Each local machine $m$ randomly chooses a mini-batch $I_m$, so totally
    $\sum_{m=1}^{g} |I_m|$ of the dual variables $x$ are updated at the $k$ iteration.
    4. Each local machine updates $x_{i,m}, i \in I_m$.
    $x_{i,m}^{(t)} \leftarrow prox(r_{i,m}^{(t)} | \frac{f_{i,m}^*}{\rho\eta_{Z,I}})$ for each $i \in I_m$, where $r_{i,m}^{(t)}$ is the $i_{th}$ element of $r_{I_m}^{(t)}$, and
    $r_{I_m}^{(t)} = x_{I_m}^{(t-1)} + (Z_{(m),I_m}^T/(\rho\eta_{Z,I})) \times \{w_{(m)}^{(t-1)} - \rho(Z_{(m)}x_{(m)}^{(t-1)} + B_{(m)}y^{(t)})\}$
    5. Each local machine updates $w_{(m)}^{(t)}$
    $w_{(m)}^{(t)} = w_{(m)}^{(t-1)} - \gamma\rho\{n(Z_{(m)}x_{(m)}^{(t)} + B_{(m)}y^{(t)}) - (n - n/K)(Z_{(m)}x_{(m)}^{(t-1)} + B_{(m)}y^{(t-1)})\}$
    and then computes $c_{(m)}^{(t)}$ and send it to the master.
    $c_{(m)}^{(t)} = \frac{B_{(m)}^T}{\rho\eta_B}\{w_{(m)}^{(t)} - \rho(Z_{(m)}x_{(m)}^{(t)} + B_{(m)}y^{(t)})\}$
```

Fig. 2. Algorithm for operating on dual domain

Communication-Efficient Online Semi-Supervised Learning in Client-Server Settings

Our framework is designed based on the above considerations. It can be decomposed into several components that drive different functionalities. On the client side, we per- form data triage by selecting instances from a candidate pool, where the selection criterion is controlled by the server. On the server side, an online semi-supervised learning algorithm is employed to handle unlabeled submissions. The key is to maintain two learners—a graph-based semi-supervised model and a linear classifier—and let them collaborate to exploit unlabeled data. Specifically, incoming instances are added to the training set of the first learner, which is represented by a graph. The nodes of the graph are instances, and the edges between nodes reflect the similarity between the corresponding instances. Then, the first learner predicts labels for all unlabeled instances in the graph, and randomly samples an instance according to the confidence of its predictions in order to teach the second learner. The second learner updates its hypothesis, and delivers a new selection criterion to the client. At any time, the second learner can be used as a standalone model for predicting new test data.

While different machine learning algorithms can be used as a part of this framework, some techniques lend themselves to our problem setting better than others. In this work, we use the harmonic solution (HS) as the first learner and the soft confidence-weighted classifier (SCW) as the second leaner. Our choice offers several advantages. First, SCW is simple, fast and enjoys state-of-the-art performance on classification. Second, SCW performs a conservative update especially with noisy labels. Third, SCW can be parameterized by a weight vector and a covariance matrix, allowing the server to deliver the selection criterion to the client with a low communication cost. In this work, we simply transmit the weight vector of SCW to the client. On the other hand, HS nicely complements SCW by providing feedback using the data manifold. It can leverage the similarities between instances, which is something that SCW overlooks, to deter- mine labels of unlabeled data. By peering these two models together, we enjoy the best of both worlds, efficient learning and simple parameterization due to SCW, and the ability to exploit manifold information disclosed by unlabeled examples due to HS. Moreover, SCW and HS can be incorporated into a single optimization problem.

One may find it is debatable whether a two-learner structure is really a preferable choice comparing to a single learner. For example, one of the alternatives is to train a linear classifier using its own predicted labels without leveraging data manifold information. Unfortunately, such an idea is not effective according to our experiments. Sometimes, the results are even worse than not using any unlabeled data. The reason is twofold. First, a single unlabeled instance can hardly provide any useful information. Second, most of the online linear classifiers only return a single hypothesis on each round, precluding any other possible hypotheses. Hence, some previous work employed Bayesian methods to update a (posterior) distribution over the hypothesis. Unfortunately, the posterior is often complicated. It is not known how to perform the update analytically. Therefore, the learning process can be easily misled and stuck in a wrong direction. Another alternative is to use a graph-based method solely. However, due to the nonparametric nature of graph-based methods, it is not straightforward to deliver the server's model to clients with a low communication cost (for the same reason, nonparametric methods are not favorable in our problem setting). Moreover, graph-based methods are also less efficient for predicting new data, as they usually involves matrix inversion. A two- learner structure, in contrast, surmounts the above problems by complementing each other's drawbacks. The choice of two learners with different underlying mechanisms is a key for good performance.

If we define the communication cost as the total number of vectors in R^d transmitted over the network, then a straight- forward implementation of our proposed framework incurs a cost of at most

$$\underbrace{l + \lfloor \frac{v-l}{q} \rfloor \omega + \min\left((v-l) \bmod q, \omega\right)}_{\text{client to server}} + \underbrace{\lfloor \frac{v-l}{q} \rfloor}_{\text{server to client}}.$$

Parallel least-squares policy iteration

We propose the parallel least-squares policy iteration to handle the large-scale learning problem. The setting is that there are M workers (cores) available (either multiple machines or multiple cores on a single machine) for computations. To fully exploit the available computational resources, each worker m collects samples and runs by itself, and then updates its estimated A^{-1}_m, b_m, and θ_m. At some point during learning, it communicates the learned θm with other workers.

The algorithm is shown in Algorithm 1 in [b3]. For every outer iteration t, each worker m individually collects samples by following e-greedy over K episodes. When collecting samples, each worker also incrementally updates $\widehat{A}_m$ and $\widehat{b}_m$. After conducting K episodes of learning, each worker also reuses the samples collected at previous iteration to updates $\widehat{A}_m$ and $\widehat{b}_m$. Then, each worker updates θm and sends it to the master. The master then averages the models to obtain the consensus z. and broadcasts it to all the workers. The workers then update the policy with the new consensus and proceed to the next iteration. After T iterations, parallel-LSPI outputs the most recent consensus z_T.

In parallel LSPI, each worker m communicates to the master only after updating its estimator θm, which occurs when it has executed sufficient number of episodes. This is the strategy that balances between communication overheads and required iterations to the optimal solution. If the algorithm dictates each worker to communicate right after every episode, communications overheads becomes heavy. If each worker independently runs the episodes during training and the parameters are only averaged at the end, the communication overhead would be minimized but it may not lead to satisfactory results as the information from other workers is completely ignored during training. Thus, a better strategy is to strike a balance between the two extreme. Compared to the related work of parallel TD that only allows small K (roughly K < 5), the proposed parallel LSPI can reduce communication cost by allowing the workers to run sufficient amount of episodes before mixing. Note that the underlying core of LSPI is least-squares temporal difference learning ($LSTD_{-}Q$), which is naturally a batch method in contrast to TD learning. Therefore, it does not need to update θ for every transition as TD learning does. Thus, parallel LSPI naturally enjoys the benefit of parallelization without the burden of frequent communication.

ANALYSIS

Here we analyze the sample complexity of the proposed method. As for non-parallel LSPI did, the analysis is first performed on a version of LSTD called *pathwise-LSTD* for policy evaluation. It analyzes LSTD at the states along a sampled trajectory following a given policy. As there are *M* workers (and *M* trajectories) with parameter averaging conducted in our case, we have to analyze the averaged estimated parameters from the trajectories. Then, one may generalize the analysis over entire state space under certain condition and derive the finite-sample bound of parallel LSPI in turn.

Thus, here we focus on *parallel pathwise-LSTD* as the insight on the sample complexity can already be seen in this step.

Before explaining pathwise-LSTD, let us first describe the notations. As an MDP is reduced to a Markov chain given a policy π, let $(X_1, X_2,...,X_n)$ be a trajectory of size n generated by the Markov chain. With abuse of notation, we denote $\Phi = [\varphi(X_1)^T;...;\varphi(X_n)^T]$ as the feature matrix defined along the trajectory. The estimated value function is thus constrained on the feature space F = $\{\Phi\vartheta,\ \vartheta \in R^d\}$. Pathwise-LSTD takes the feature matrix Φ generated by a trajectory following as input. It builds the empirical transition matrix $P : P_{ij} = \mathbb{I}\{j = i + 1, j \neq n\}$, and sets the quantities $A = \Phi^T(I - rP)\,\Phi$, and $b = \Phi^T r$. It then outputs the solution $\vartheta = A^+ b$ with minimum norm, where A^+ represents the Moore-Penrose pseudo-inverse of A.

Let us denote v as the value function and its estimated one along the trajectory, and $\{X_t\}_{t=1}^n$ $\|f\|_n^2 = \frac{1}{n}\sum_{t=1}^{n} f(X_t)^2$ as the empirical norm. Moreover, let V_{max} represent the feature space, and v be the smallest positive eigenvalue of the maximum of the value function, Π be the projection to the feature space, and v be the smallest positive eigenvalue of Gram matrix $\Phi^T\Phi/n$. From Theorem 1, $\|v - \widehat{v}\|_n$ is bounded as

$$\begin{aligned} \|v - \widehat{v}\|_n \leq & \frac{1}{\sqrt{1-\gamma^2}}\|v - \widehat{\Pi}v\|_n \\ & + \frac{1}{1\quad\gamma}[\gamma V_{max} L\sqrt{\frac{d}{\nu}}(\sqrt{\frac{8log(2d/\delta)}{n}} + \frac{1}{n})], \end{aligned} \tag{12}$$

with high probability $1 - \delta$.

For parallel pathwise-LSTD, denote $(X_{1,m}, X_{2,m}, ..., X_{n,m})$ as the m_{th} trajectory of the Markov chain induced by a policy π, $\Phi_m = [\varphi(X_{1,m})^T;...;\varphi(X_{n,m})^T]$ as the corresponding feature matrix, and v_m as the smallest positive eigenvalue of the corresponding sample based Gram matrix $v_m,\ \widehat{v}_m = \Phi_m\widehat{\theta}_m$ of features. Moreover, let $\widehat{\theta}_m$ $\|v' - \Phi'\widehat{\theta}\|_n$ represent the pathwise solution of the trajectory m, andrepresent the value function $\|v' - \Phi'\widehat{\theta}\|_n$ and the estimated one at the states along trajectory m respectively. Since parallel LSPI conducts parameter averaging, we are interested in the sample complexity associated with the averaged estimator $\widehat{\theta} = \Sigma_m\widehat{\theta}_m/M$. Let π'represent the new policy at the next iteration of parallel LSPI and$X_1', X_2', \ldots, X_n'$ be the trajectory of the Markov chain following π with Φ being the corresponding feature matrix. Then, we want to analyze the quantity as it would give us insight on the sample complexity. To estimate the upper bound, we make the following assumption that connects the performance of θ_m evaluated at the new trajectory $\{X_t'\}_{t=1}^n$ to the one evaluated at the original$\{X_{t,m}\}_{t=1}^n$ where θ_m is estimated from.

Assumption *There exists a constant c such that the empirical norm of the difference between value function v and the one implied by* θ_m *evaluated at the trajectory* $\{X'_t\}_{t=1}^n$ *is upper bounded as* $\|v' - \Phi'\theta_m\|_n^2 \leq c^2\|v_m - \Phi_m\theta_m\|_n^2$.

Notice that the empirical norms on both sides of the inequality are evaluated at different trajectories; the left hand side is on $\{X'_t\}_{t=1}^n$, while the right hand side is on $\{X_{t,m}\}_{t=1}^n$ for an *m*. Using the assumption, we can bound, $\|v' - \Phi'\theta\|_n$ which measures the performance of the averaged estimator.

Proposition *Following the above assumptions, if* $(v' - \Phi'\theta_m)$ *of each m is near orthogonal, we have*

$$\|v' - \Phi'\widehat{\theta}\|_n \leq \frac{c}{\sqrt{M(1-\gamma^2)}} Max_m\{\|v_m - \Pi_m v_m\|_n\} + \\ + \frac{c}{1-\gamma}[\gamma V_{max} L\sqrt{\frac{d}{\nu_{min}}}(\sqrt{\frac{8log(2d/\delta)}{Mn}} + \frac{1}{n\sqrt{M}})], \quad (13)$$

where ν_{min} *represents the smallest of the* ν_m, *and* $Max_m\{\|v_m - \Pi_m v_m\|_n\}$ *represents the largest* $\|v_m - \Pi_m v_m\|_n$ *of the terms. If* $(v' - \Phi'\theta_m)$ *are highly correlated, we have*

$$\|v' - \Phi'\widehat{\theta}\|_n \leq \frac{1}{\sqrt{1-\gamma^2}} Max_m\{\|v_m - \Pi_m v_m\|_n\} \\ + \frac{1}{1-\gamma}[\gamma V_{max} L\sqrt{\frac{d}{\nu_{min}}}(\sqrt{\frac{8log(2d/\delta)}{n}} + \frac{1}{n})]. \quad (14)$$

The proposition suggests that, for the ideal case, parallel LSTD can estimate the value function at an improved rate O(1/√*Mn*) of each worker comparing to the original rate of O(1/√*n*). This implies that the effectiveness of process in which *M* workers collect some *n/M* samples in parallel is comparable to a single worker collecting *n* number of samples. Yet, the sampling is conducted in a distributed fashion in parallel LSTD as compared to standard LSTD. At the other extreme, for the worst case scenario, parallel LSTD has the same sample rate, O(1/√*n*), for each worker as that of standard LSTD. Yet, each worker individually collects *n* samples meaning that the total samples in parallel LSTD are *M* times larger than that of the non-paralleled one.

Proof: First, let us rewrite $\|v' - \Phi'\widehat{\theta}\|_n^2$ as $\frac{1}{M^2}\|\sum_{m=1}^{M}(v' - \Phi'\widehat{\theta}_m)\|_n^2$. Suppose, for each m, $(v' - \Phi'\widehat{\theta}_m)$ is nearly mutual orthogonal. Then, we have

$$\frac{1}{M^2}\|\sum_{m=1}^{M}(v' - \Phi'\widehat{\theta}_m)\|_n^2 \approx \frac{1}{M^2}\sum_{m=1}^{M}\|(v' - \Phi'\widehat{\theta}_m)\|_n^2$$
$$\leq \frac{c^2}{M^2}\sum_{m=1}^{M}\|(v_m - \Phi_m\widehat{\theta}_m)\|_n^2 \leq \frac{c^2}{M}G^2, \quad (15)$$

where

$$G = \frac{1}{\sqrt{1-\gamma^2}} Max_m\{\|v_m - \Pi_m v_m\|_n\} +$$
$$+ \frac{1}{1-\gamma}[\gamma V_{max} L\sqrt{\frac{d}{\nu_{min}}}(\sqrt{\frac{8log(2d/\delta)}{n}} + \frac{1}{n})]. \quad (16)$$

Taking square root on both sides gives us the result.

In contrast, if all $(v' - \Phi'\widehat{\theta}_m)$ are highly correlated, we have

$$\|v' - \Phi'\widehat{\theta}\|_n^2 \approx \frac{1}{M^2}\|\sum_{k=1}^{M}(v_m - \Phi'\widehat{\theta}_m)\|_n^2 = \|(v_m - \Phi'\widehat{\theta}_m)\|_n^2$$

which reduces to the case of a single worker.

To achieve better performance for parallel LSPI, according to the proposition, we should make $(v' - \Phi'\widehat{\theta}_m)$ as less correlated to each other as possible.

However, $(v' - \Phi'\theta_m)$ is unknown in advance. To deal with the issue, a heuristic is used to enforce each worker to take random actions more when collecting samples. Since e-greedy is adopted here, a larger that encourages more exploration (i.e. randomly choosing an action) will bring us closer to the goal. If gets close to zero, the randomness would mostly come from the transitions *P* of the process. In this case, performance of parallel LSPI may not achieve significant speedup, depending on the underlying MDP and the feature space. Note that the degree of correlation between θ_m is not equivalent to the degree of the correlation between $(v' - \Phi'\theta_m)$ Still, our experiments reveal that such heuristic does have positive effect on the performance.

Results and Discussion:

Speeding up ADMM

We compare our methods with standard ADMM and distributed SDCA on several datasets. We denote our first method which performs ADMM on primal objective (1) with sampling as ADMM-P, and we denote the second method which performs distributed SDCA-ADMM on dual objective (9) as ADMM-D.

The datasets are binary classification. For dataset: delta, gamma, and ocr are available on http://largescale.ml.tu-berlin.de/about, while for dataset: mnist, epsilon, and rcv1 are available on http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/. Table 1 shows their statistics. For delta, gamma, and ocr, because only the labels of original training data are available, we split each of them by 80% as training set and the remain 20% as testing set. For mnist, we choose classifying digit 4 against digit 7 as the recognition goal and also split the data into 80/20 split. For epsilon, we use the pre-defined training/testing split. For rcv1, the ratio of original training to testing data size is much less than one, so we re-split the data into 80/20 split. The data are further distributed on four machines in our workstation. The batch size |I| is set to 100 (i.e each of four machines uses 25 samples at a time) for both SDCA and ADMM-D on all the datasets except rcv1 where the batch size $|I|$ is set to 1000.

Data	number of samples	dimension	Data	number of samples	dimension
delta	500,000	500	ocr	3,500,000	1,156
gamma	500,000	500	epsilon	500,000	2,000
mnist47	1,634,445	784	rcv1	697,641	47,236

Table 1. Dataset Statistics

There are some parameters needed to be specified in ADMM-D: $\rho,\gamma,\eta_{\{Z,I\}}$, and η_B. For γ, we set it as $\gamma=1/n$. For η_B, due to the choice of transform B, simple calculation would show that it should be at least gp (i.e. the value no less than the multiplication of number of machines g and feature dimensions p). For $\eta_{\{Z,I\}}$, it should be larger than the largest eigenvalue of $Z^T_I Z_I$ associated with the mini-batch I. But it is time consuming to compute it for each mini-batch at the augmented feature space. Instead, before running the optimization, we just sample a mini-batch, say I, and calculate the largest eigenvalue of $Z^T_I Z_I$ at the original feature space. Denote the computed value as $\eta_{\{tmp\}}$. We set every $\eta_{\{Z,I\}}$ to the same value: $\eta_{\{Z,I\}} = \theta$, where θ= 5 x 10^d with the smallest power d such that$\theta>\eta_{\{tmp\}}$. For ρ, it is set to $10^{d'}$ with d' chosen to satisfy ρ x $\eta_{\{Z,I\} =}$ 5. We found the heuristic work well.

The parameters for standard distributed ADMM is set to the default setting, while for the ADMM-P, we set the additional parameter k to 1.5, which means that samples used is increased by 1.5 times at subsequent iterations (we did not find any value that significantly leads to better results.). Since the objective of each method is not the same (i.e. for ADMM-D, the objective is shown on (9); for standard ADMM and ADMM-P, the objective is shown on (1); for parallel-SDCA, the objective is hinge loss with L2 regularization without any constraint as compared to the others.), we tune the regularization parameter C such that each method can achieve to the best accuracy on each dataset.

Fig. 2 shows the results, which are about training accuracy vs. time. Each curve represents a different method. For ADMM or ADMM-P, each point on a curve is the classification result of a model at a corresponding iteration, while for distributed SDCA or ADMM-D, each point is the result at every the-number-of-batches iterations (which is roughly equivalent to a full pass of data.). We run the distributed SDCA and ADMM-D ten times for each dataset so the results are the averaged ones.

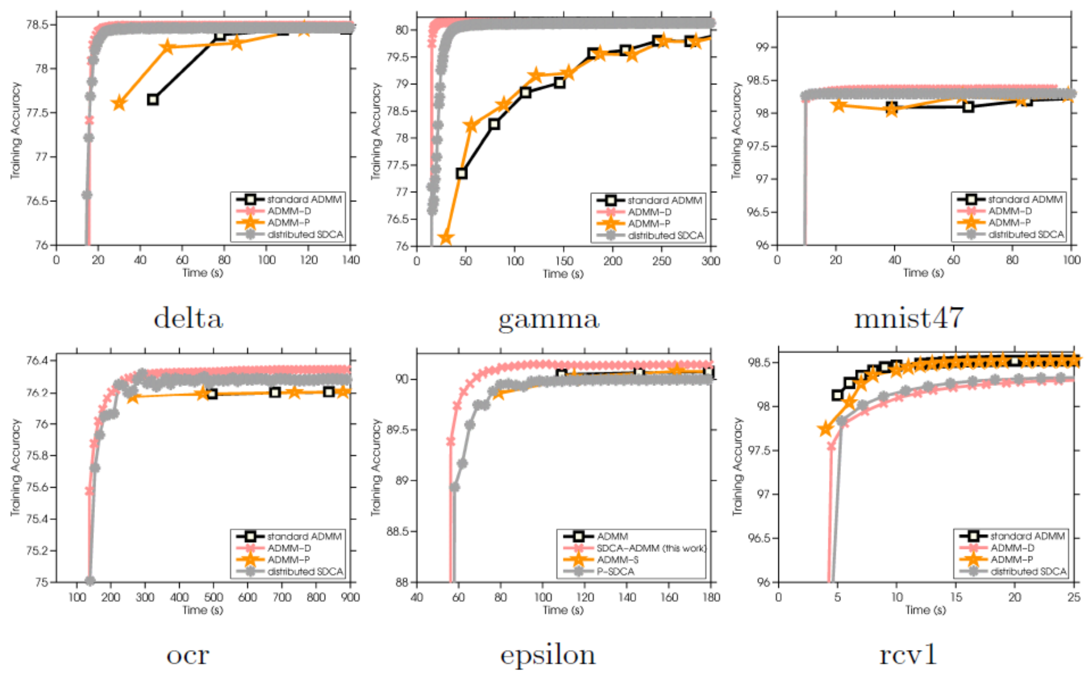


Fig 2. Training accuracy vs. time for each method on different datasets.

From the figures, we see that ADMM-D converges the fastest on most of the datasets except rcv1 among the methods. The figure also shows that ADMM-D and distributed SDCA outperform standard ADMM and ADMM-P on most of the cases, while ADMM-D is better than distributed SDCA on gamma and epsilon. On other data, ADMM-D is comparable or slightly better than distributed SDCA. This indicates that ADMM-D (distributed SDCA-ADMM) can enjoy the merits of SDCA and ADMM and outperform them as a consequence. The figure also shows ADMM-P is at least comparable to standard ADMM. On rcv1, standard ADMM seems to be better than ADMM-D and distributed SDCA. Note that the feature dimensions of rcv1 data is much larger than the other datasets. As the dimensions increases, computations such as matrix-vector product also scales with the dimensions. For standard ADMM, the algorithm follows a strategy used in LIBSVM, where it maintains and updates a active set such that the dual variables associated with samples outside the active set do need to be updated anymore. This means that the computational time is decreased through iterations of standard ADMM, which alleviates the suffer of high dimension. While in SDCA or ADMM-D, at each iteration, it still samples a pre-determined batch size of data. Thus, an interesting future work is exploiting the active set strategy into distributed SDCA and ADMM-D. To summarize, ADMM-D is a effective method on medium feature size data. If frequent communication is admissible (as in our setup), we suggest to use ADMM-D, otherwise use ADMM-P.

We propose sampling-based ADMM approaches for learning from distributed data. We integrate the idea from stochastic gradient descent into ADMM. Our first method uses subset of data on early rounds of communication, which can reduce the cost on early stage while enjoy the similar convergence rate as the standard one. We also transform the primal objective into the approximated dual form and propose a distributed variant of the recently proposed SDCA-ADMM to solve it.

Communication-Efficient Online Semi-Supervised Learning i

Experiments were conducted on seven data sets downloaded from either the UCI ML repository (wearable, skin) or the LIBSVM website (mushroom, mnist, webspam, gisette, ijcnn1). The motion recognition data set wearable and digit recognition data set mnist were converted into a set of bi- nary problems, respectively, where each class is discriminated against every other class. Totally, we produced 10 problems from wearable and 45 from mnist. For each data set, we balanced the number of instances of each class and linearly rescaled the feature values into the range [−1, 1].

We evaluated the algorithms using a set of trials with different partitions of the training and test data. In each trial, we randomly held out half of the data for testing; all instances in the test set were labeled by the algorithms. The remaining data was used for training, of which only a small amount was labeled. Both training and test sets were class- balanced. Next, we randomly permuted the training data and kept labeled data always at the beginning. All algorithms were then incrementally trained with the same permutation in each trial. For evaluation, we paused the training at regular intervals, computed the output hypothesis so far, and calculated its test accuracy.

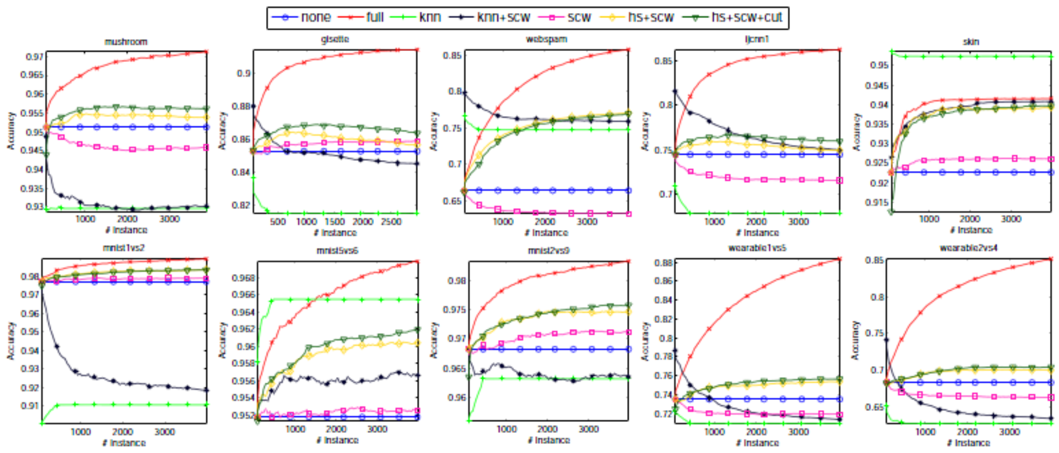


Fig. 3. Test accuracy of different models on the server. The x-axis represents the number of unlabeled instances on the client. The origin corresponds to the point where the initial 2% of the data has been labeled and learned and the first unlabeled instance comes in. The client randomly selected 10 instances from every 50 instances. `full` is an idealized approach in which an oracle labels all selected instances. `none` does not upload any unlabeled instance to the server, so the corresponding test accuracy is constant. A tournament result between algorithms on all problems of mnist and wearable are presented in Tables I(a) to I(b).

Comparison of Server's Model

Fig 3. shows the results. It can be observed that proposed hs+scw and hs+scw+cut enjoy superior performance on 8 out of 10 problems comparing to other partial label competitors. On 45 mnist problems, hs+scw and hs+scw+cut yielded on average 0.966 and 0.971 accuracy, respectively. On 20 wearable problems, hs+scw and hs+scw+cut gave 0.699 and 0.714 accuracy, respectively. They are consistently better than the single-learner counterpart scw on all data sets. This indicates the effectiveness of leveraging manifold information of the graph. In fact, on webspam, ijcnn1 and wearable, scw is even worse than none. On webspam, its test accuracy starts with 0.658, decreasing over time and finally yielded 0.637. This is due to the fact that scw completely relies on its own prediction for learning. When the labeling rate is small, the initial hypothesis constructed by labeled data may not be accurate enough. As a consequence, the prediction of scw on

Comparison of Selection Strategy

Fixing the model on the server as hs+scw+cut, we study the following strategies on the client side.

All, All unlabeled instances are uploaded without selection. This incurs 5x the communication costs versus other approaches.

rand Randomly selects instances for uploading.

certain The most certain instances according to the current server model

uncertain The most uncertain instances are uploaded.

submod Selection is done by optimizing the submodular function described in Section VI. It simultaneously considers the uncertainty and redundancy.

TABLE 2 PERFORMANCE COMPARISON OF DIFFERENT SELECTION STRATEGIES. THE VALUE IN THE TABLE DENOTES THE NUMBER OF TIMES THAT THE ROW ALGORITHM ACHIEVED SIGNIFICANTLY BETTER ACCURACY THAN THE COLUMN ALGORITHM UNDER THE t-TEST WITH $p = 0.05$.

(a) Different server's models on mnist (45 problems).

	none	full	knn	knn+scw	scw	hs+scw	hs+scw+cut
none	-	0	12	9	6	0	0
full	45	-	39	35	35	30	34
knn	20	0	-	10	13	4	3
knn+scw	13	0	3	-	10	0	1
scw	2	0	3	2	-	1	1
hs+scw	31	1	15	22	24	-	3
hs+scw+cut	30	1	15	21	23	2	-

(b) Different server's models on wearable (10 problems).

	none	full	knn	knn+scw	scw	hs+scw	hs+scw+cut
none	-	0	6	6	5	0	0
full	10	-	10	10	10	10	10
knn	0	0	-	1	0	0	0
knn+scw	0	0	2	-	0	0	0
scw	0	0	3	2	-	0	0
hs+scw	6	0	9	8	8	-	1
hs+scw+cut	6	0	8	8	7	0	-

(c) Different selection strategies on mnist (45 problems).

	none	full	all	rand	certain	uncertain	submod
none	-	0	0	9	6	0	0
full	45	-	45	45	45	45	45
all	30	0	-	2	27	2	1
rand	30	0	1	-	1	0	0
certain	34	0	9	2	-	0	0
uncertain	35	0	15	22	24	-	0
submod	38	0	17	25	25	6	-

(d) Different selection strategies on wearable (10 problems).

	none	full	all	rand	certain	uncertain	submod
none	-	0	0	0	0	0	0
full	10	-	10	10	10	10	10
all	3	0	-	0	1	1	0
rand	3	0	0	-	1	0	0
certain	3	0	0	0	-	0	0
uncertain	5	0	2	2	2	-	0
submod	6	0	3	2	2	0	-

The results are shown in Table 2. It is interesting to see that All, which transmits all unlabeled data, does not lead to a better performance. In fact, on mnist, mushroom, and gisette, All yields worse test accuracy compared to selective transmission. This confirms the intuition that not all unlabeled instances are useful. It also suggests the necessity of using a selective sampling strategy on the client. Not only the communication costs can be saved, but also a better model might be learned.

Parallel least-squares policy iteration

We conduct the experiments on two domains. One is the discrete-time four-dimensional queuing network. Figure 4 illustrates the network, which includes four queues, each with buffer size B. Here server 1 can only serve queue 1 or 4, and server 2 can only serves queue 2 or 3 one at a time but not simultaneously. Each server can only handle one customer at a time at most. Moreover, neither server can be idle. Let the tuple {a1,a2,a3,a4} represents an action combination which the servers take by considering conditions in the queues {q1,q2,q3,q4}, where ai = {1,0} indicates whether qi is currently being served or not. Then, there are total four actions {1,1,0,0} {0,1,0,1} {1,0,1,0} {0,0,1,1} the servers can take. As the result, the number of states is (1+B)4×4, which means that a modest B will result in a huge state space.

The dynamics of the network are defined by the rate parameters μ1,μ3,d1,d2,d3,d4 ∈ (0,1), all follow Bernoulli distribution. μ1 and μ3 are coming rates of new customers. At each time step, with probability μi, a new customer comes to queue i. di is defined as follows: if ai = 1, which indicates queue i is being served, the server would succeed in handling a customer with probability di before the next time step, and fail with probability 1 − di. Starting with empty queues, each

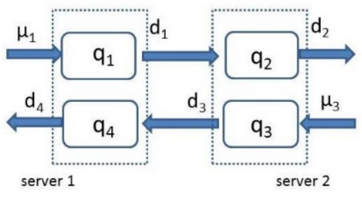


Fig. 4. The discrete-time four dimensional queuing network. Customers can arrive at q1 or q3. The customer that is served and finished by q1/q3 is then referred to q2/q4.

episode spans a fixed number of time steps, T. The goal is to minimize the average of total waiting (unserved) customers in the network during an episode. The loss for a state-action pair is defined as l(s,a) = l(s) = |X|, which is the total number of unserved customers in all the queues. After an episode, the network is reset to empty and a new episode begins.

We consider four types of networks:

1) μ1 = 0.5,μ3 = 0.5,d1 = 0.5,d2 = 0.8,d3 = 0.8,d4 = 0.5, episode duration T = 50, and buffer size B = 5, which results in 5,184 state-action pairs.

2) μ1 = 0.5,μ3 = 0.8,d1 = 0.5,d2 = 0.1,d3 = 0.8,d4 = 0.8, episode duration T = 100, and buffer size B = 10, which results in 58,564 state-action pairs.

3) μ1 = 0.4,μ3 = 0.4,d1 = 0.5,d2 = 0.8,d3 = 0.3,d4 = 0.3, episode duration T = 200, and buffer size B = 15, which results in 262,144 state-action pairs.

4) μ1 = 0.4,μ3 = 0.4,d1 = 0.4,d2 = 0.8,d3 = 0.8,d4 = 0.4, episode duration T = 200, and buffer size B = 15, which results in 262,144 state-action pairs.

We design 340-dimensional sparse binary feature for type 1 network and 1048-dimensional features for the others. In our design, only two entries in the feature are non-zeros for each state-action pair.

Another domain is the persistent search and track. The scenario is that there are three Unmanned Aerial Vehicles (UAV) to corporate for a mission. There are three available actions for each UAV: {advance,retreat,loiter}, resulting in 27 total possible action combinations. The current state of a UAV is described by : location, fuel, actuator status, and camera status. The goal is to fly to the target site and perform surveillance, while ensuring that there is a UAV with a working actuator loitering at the intermediary site to transfer the information of the targets to the base.

We modify the scenario because the reported performance of LSPI is not good in the original setting. Each UAV starts from the base with 6 units of fuels. The camera and actuator of each UAV can may with a 3% probability at each time step. The camera cannot function under failed actuator, so a UAV with a failed actuator cannot perform surveillance. Yet, a UAV can perform communication even its camera malfunctions. A successful surveillance mission must have at least one UAV with working actuator at the intermediary site, and at least one UAV with working actuator and camera at the surveillance site. At each time step, each UAV loses 1 unit of fuel except when it "loiters" at the base or at the intermediary site. When a UAV "loiters" at the base, the failed camera and actuator are fixed, and the fuels are recharged fully. When a UAV with working actuator "loiters" at the intermediary site, the messages is transmitted to the base

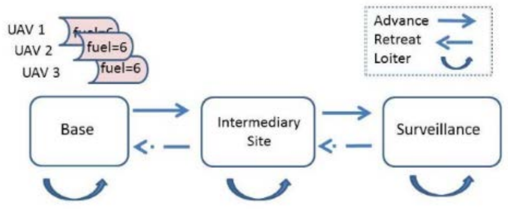


Fig. 5. Persistent search and track.

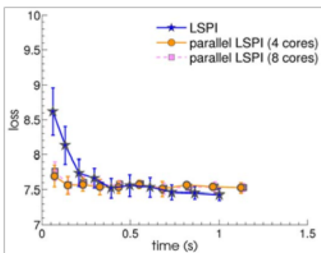
and UAV's fuel tank is recharged by 2 units (the fuel cannot exceed the capacity, which is 6 units). If a UAV "retreats" at the base, then it may "advance" to the intermediary site or "loiter" at the base with equal probability. Executing "advance" action at the surveillance site has similar effect. The reward for each state-action pair is defined as r(s,a) = 15×Icomm × Isurv−10×Icrash−(18−total remained fuels), where Icomm indicates whether there is a UAV with working actuator at the intermediary site, Isurv indicates whether there is a UAV with working actuator and camera at the surveillance site, and Icrash represents whether a UAV crashes. If a UAV runs out of fuel, which means it crashes, the episode is terminated. In total, the state-action pair has the size of about 1.6×106, and about 2000-dimensional sparse binary feature including fixed sparse representation.

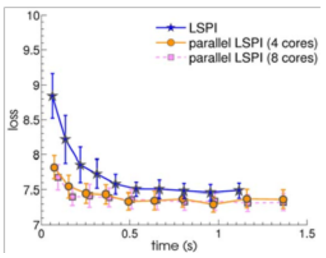
B. Setup and Results

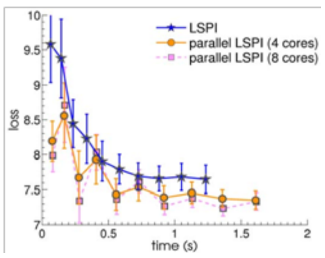
We compare parallel LSPI with standard LSPI on the two domains. Both methods are implemented in C and the communication in parallel LSPI is implemented with MPI. For parallel LSPI, we report the performance of M = 4 and M = 8 workers. In our implementation, parallel LSPI uses single core machines, yet the shared-memory architecture (i.e. multi-core on a single machine) is also applicable. For the queuing network domain, LSPI and parallel-LSPI update their policies every 100 episodes, and both of them terminate after learning 1000 episodes, which corresponds to K = 100 and T = 10 in the algorithm. For persistent search and track, both LSPI and parallel-LSPI update their policies every 1000 episodes, and terminate after 10,000 episodes. We set $\gamma = 0.95$ for both domains. All the experiments are repeated 50 runs with the averaged results and standard deviations reported.

We also try differentϵ for ϵ-greedy policy. Higher means each worker takes random action with higher probability, which could reduce the correlation between workers. The results for the queuing network are shown in Figure 6, and results for persistent search and track are shown in Figure 7. The learned parameter θ is recorded when it is updated, so each point on the line represents the performance of learned parameter at the end of an iteration of the corresponding algorithm. For parallel LSPI, we record the consensus as the the learned parameter. The performance of a learned parameter in the queuing network domain is measured by the average of losses (where loss of an episode is defined as the average of all waiting customers in the network during an episode) over additional 500 episodes, which are conducted by following the deterministic policy implied by the learned parameter. In persistent search and track domain, the performance is measured by cumulated discounted rewards, with the same evaluation procedure as the queueing network.

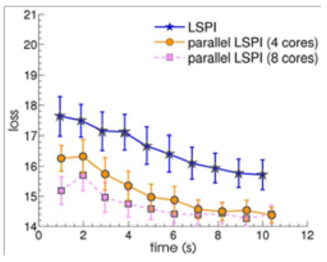
Figure 6 suggests for queueing network, higher e yields better parallelization since the correlation between workers is smaller. At higher (left column), which encourages taking random action more during learning, parallel LSPI can significantly accelerate the learning process comparing to taking action more greedily with respect to the current estimated state action value (right column). For parallel-LSPI with 0.7 or 0.5greedy, after running three or four iterations, it already reaches the point where the standard LSPI needs to take ten iterations or more. For 0.2-greedy in network 3 and 4, parallel LSPI has no advantage while consumes more computation resources than non-parallelized one each iteration. For persistent and search domain, parallel-LSPI with $\epsilon = 0.1$-greedy already

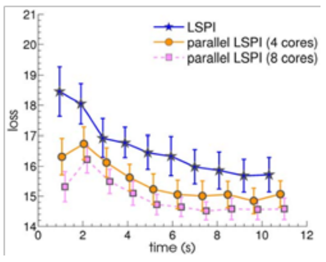


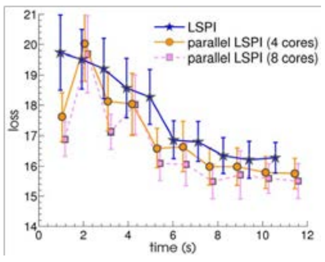




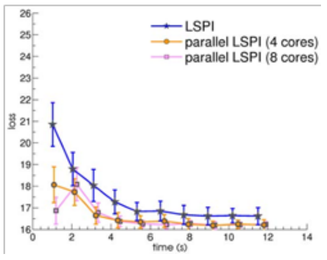
(a) 0.7-greedy in type 1 network (b) 0.5-greedy in type 1 network (c) 0.2-greedy in type 1 network

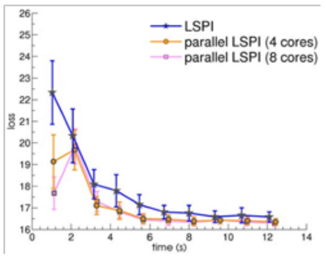


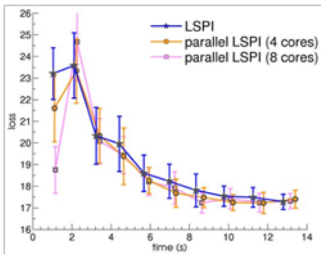




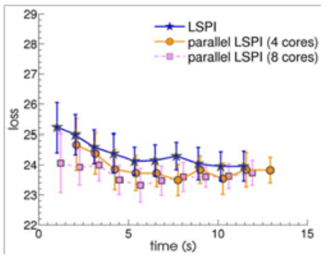
(d) 0.7-greedy in type 2 network (e) 0.5-greedy in type 2 network(f) 0.2-greedy in type 2 network

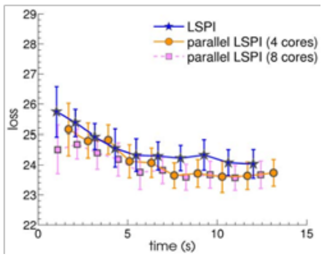


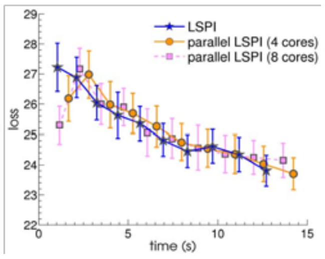




(g) 0.7-greedy in type 3 network (h) 0.5-greedy in type 3 network(i) 0.2-greedy in type 3 network





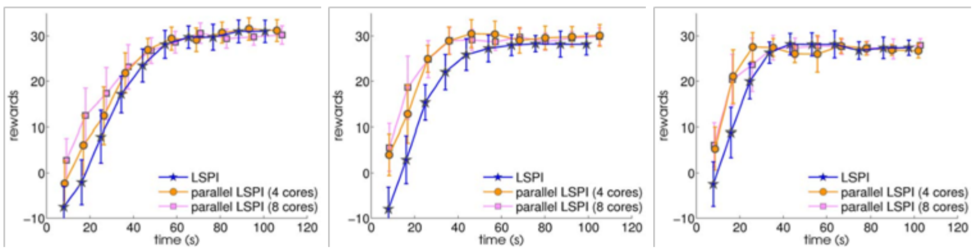


(j) 0.7-greedy in type 4 network (k) 0.5-greedy in type 4 network(l) 0.2-greedy in type 4 network

Fig. 6. Losses of the learned parameter versus learning time. Each row corresponds to a network, while each column corresponds to different. Star marker represents LSPI, circle marker represents parallel LSPI with four workers, and square marker represents parallel LSPI with eight workers. The 0.5 s.e. error bars are also plotted.

achieves significant speedup than its non-parallel counterpart, increasing can accelerate further but not much more. This reflects the limitation of the proposed heuristic, yet parallelLSPI still shows the benefit of parallelization. The overhead of parallelization can also be seen on the figures. We can see that parallel LSPI requires slightly more time to finish the same amount of iterations than standard LSPI does due to communication overhead. Yet, this overhead is tolerable since parallel LSPI usually reaches at the same level of performance as standard LSPI with much fewer iterations.

From the figures, we also observe that the advantage of parallelization decreases as the number of workers increases (4 workers vs. 8 workers). The acceleration by doubling the workers is incremental in most of the cases. A possible



(a) 0.1-greedy (b) 0.2-greedy (c) 0.35-greedy

Fig. 7. Rewards of the learned parameter in persistent search and track domain versus learning time. The 0.5 s.e. error bars are also plotted.

A possible explanation is that the correlation between workers is likely to increase with more workers being added. Similar to the parallel TD, the degree of parallelization of our method still has some room for improvement. This limitation is not explicitly implied in our analysis since we just show the ideal case and the worst case. That says, the connection between the degree of possible parallelization and the properties of underlying MDPs and feature space needs to be further explored. We leave it as a future work.

List of Publications and Significant Collaborations that resulted from your AOARD supported project: In standard format showing authors, title, journal, issue, pages, and date, for each category list the following:

a) papers published in peer-reviewed journals,

b) papers published in peer-reviewed conference proceedings, 1 Han Xiao , Shou-De Lin , Mi-Yen Yeh , Phillip B. Gibbons and Claudia Eckert ”Learning better while sending less: Communication-efficient online semi-supervised learning in client-server settings” DSAA 2015

2 Jun-Kun Wang, Shou-De Lin “Efficient Sampling-based ADMM for Distributed Data”, DSAA 2016.

3 Jun-Kun Wang, Shou-De Lin “Parallel Least-Squares Policy Iteration”, DSAA 2016.

c) papers published in non-peer-reviewed journals and conference proceedings,

d) conference presentations without papers,

e) manuscripts submitted but not yet published, and

f) provide a list any interactions with industry or with Air Force Research Laboratory scientists or significant collaborations that resulted from this work.

Attachments: Publications a), b) and c) listed above if possible.

As attached